APPENDIX F

EXPRESS MAIL LABEL
NO.: EL386270487US

Introduction

Appendix F will be understood and appreciated from the following detailed description, taken in conjuction with the drawings in which:

Fig. 64 is a simplified block diagram illustrating inputs and outputs of a task_trigger_handler, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 65 is a simplified flow chart illustrating the function of a task_trigger_handler, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 66 is a simplified illustration of the run length structure of a task_trigger_handler where the map region is the bounding rectangle enlarged by two pixels on either side, constructed and operative in accordance with a preferred embodiment of the present invention;

Fig. 67A is a simplified illustration of the snap rectangular regions cut according to the clusters defined in Fig. 66; and

Fig. 67B is a simplified illustration of the snap rectangular windows possessing the cluster, and a rectangle with RGB data, after processing of the clusters defined in Fig. 66.

This appendix describes a task existing in the SIP, named Task_trigger_handler. The purpose of this task is a primary fast sorting of information that originally exists in separated structures in the snap channel, and combining the relevant data into a single structure which is ready for futher processing. The task consumes two data sources - data source of Color_defects and data source of Snaps. The produced data source is of the type queue of windows. The operation of the task consists of three main parts : (i) on-line clustering of the Color-defect reports, (ii) cutting the appropriate rectangular areas which contain RGB-data out of the snap data source, and (iii) packing all of the relevant information into a form of a queue of sip windows. All of these sub-parts are applied on the data accumulated so far.

Reference is now made to Figure 64 which is a block diagram of a Task_trigger_handler and its inputs of a data source of Color defects and a data source of Snaps, and producing a queue of windows as an output.

Scope

Task_trigger_handler is applied on the input data sources while they are built. The resulting output, i.e., queue of windows, is treated by the Window_test_manager, and each window is directed for further processing, according to the associated test function. Therefore, Task_trigger_handler is the first station in the post process procedure, and its main goal is an efficient gathering of associated pieces of data needed for the post process.

Basic concepts

• Color_defect report : Color_defect report is a general name for reports that enter the SIP through the snap channel. These defects may be reported either by the Color Multi-Resolution Area Defect detector (COMRADD) which is described in Applicant's copending Israel Patent Application IL 131092 entitled "Optical Inspection System", filed 25 July 1999, or the Small Defect Detector (SDD), and may be associated with various types of surface defects, such as oxidations, stains, scratches, etc.

• Snap : Snap is a rectangular or modular-rectangular region that contains RGB-information. Color_defect report causes a recording of the RGB data inside a square area around it. The minimal size of the RGB- square is 48 X 48 (pixels).

• Data source : data source is one of the SIP building blocks, that carries data which is used by other SIP units. The reports that enter the SIP through the various channels are transformed into the format of data sources.

• Queue of windows :queue of pointers to Sipwin objects. Each Sipwin objects is a rectangle of a given size and location which includes a vector of Sipdata objects and a function to be executed by the window test manager.

•

Input/Output

Task _trigger_handler treats data sources that are originated in the snap channel during the scan. The input consists of two data sources: (i) data source of Color_defect reports and (ii) data source of Snap reports. The task produces a single data source - queue of windows. The specification of the consumed and produced data sources will be given in the following.

Consumed data sources

Task_trigger_handler consumes data sources of two types:

• Ds_array<Color_defect> - the source of reports to be clustered. Ds_array<Color_defect> contains an ordered (in (x,y)) vector of Color_defect reports. The Ds_array_lines keep the reports in a non-sparse way. A pointer to the beginning of each Ds_array_line is available. Each report contains its x-location (12 bits) and other specifications like type, slice of origin, and various characterizations (24 bits).

• Ds_array<Snap> - the source of the RGB data to be cut. Ds_array<Snap> contains an ordered vector of snap reports. Each snap report contains its x-location (8 bits), as produced by the hardware), and the red, green and blue values (8 bits each). In order to use the x-coordinate, it is typically converted from the 8-bit format into the regular 12-bit format. A pointer to the beginning of each ds_array_line is available.

Produced data source

Task_trigger_handler produces a single data source - a queue of windows.

Win_queue: the Win_queue is a data source of type Sipwin.

Sipwin: A general Sipwin object is a rectangular window that contains a vector of sipdata objects and a test function. The Sipwins that are produced by Task_trigger_handler contain two sip_data objects: sipdata_cluster and sipdata_RGB, and contains an appropriate test function - (presently it is an identity function).

sipdata_cluster: contains the reports that belong to the cluster.

sipdata_RGB: contains a rectangular piece of snap as an RGB image in a portable pixel map (ppm) format, and the coordinate of the top left point, relative to the global (0,0).

Method Structure

The method's spec of task trigger handler is the following:

The following steps are preferably performed:

1) clustering of Color_defect reports.

2) cutting the appropriate RGB rectangles out of the snap data.

3) packing all of the relevant information in a form of queue of sip windows.

The method is preferably performed on line.

The method may be applied only upon consumed data sources that had already been produced.

The method preferably is efficient, since it serves only as a tool for a primary fast sorting of information that prepares the data for further and deeper processing.

Reference is now made to Figure 65 which illustrates schematically the main blocks of the method, for a single loop over a bunch of scanned lines:

Clustering method

The clustering method finds clusters that consist only of points that are connected to each other. In principle the clustering is according to a given distance criterion. The input of the clustering part is an ordered set of points (color defect reports) and the distance criterion, and the output is a list of clusters and their bounding rectangles.

The clustering of points is done line by line, and uses the assumption that the color defects are ordered according to their coordinates.

For each point in the line it is determined to which (and how many) cluster/s it belongs. In the case of connectivity criterion in two dimensions, the maximal number of clusters to which a point can belong, N_max, is 2. If the distance criterion is general, N_max 4.

- N_max = 0 => a new cluster is opened with the current point.
- N_max = 1 => the point is added to the appropriate existing cluster.
- N_max = 2 => the point is added to one of the two clusters, and a merging procedure between the two clusters is performed.

The determination of N_max uses a "history buffer" which keeps the last clustered line, such that at every site which was occupied by a color_defect there is a pointer to the appropriate cluster (to which the color_defect belongs). This is use for a random access search for neighboring color_defects. There is an associated structure that keeps only pointers to occupied sites, for a more efficient update.

Each time that a certain number of lines (determined in advance) are passed, the clusters are checked to be ready. Each cluster, includes the color_defect points and the bounding rectangle.

Cutting of snaps

This part of the task, is performed by the Snap_cutter object, is responsible for the extraction of the RGB data in a given rectangular region, out of the full RGB-data existing in the data source of snaps. The Snap_cutter gets a set of sorted rectangles as an input (sorted according to the right-bottom point). These rectangles determine the regions in which the RGB data is typically filled. The outcome of this part is a set of rectangular RGB images, kept in a ppm format.

The method for cutting snap regions according to a given set of sorted rectangles is the following is performed in several steps.

• As a first step, a conversion of the set of rectangles to a run-length structure is performed. The elements in each run-length line represent the boundaries of the rectangles (with indication of location and if it is left or right boundary and a pointer to the appropriate cluster).

• As soon as the run-length structure is built, the RGB data is filled accordingly. The filling is done separately for each line, according to the following method:

1) Using the run-length data, one can determine the number of rectangles to which a segment between two consecutive run-length elements belongs.

2) A search in the data source of snaps for the pixels that belong to the segment is performed. The search uses the natural division of the data source of snaps into blocks of 16 pixels (all pixels inside such a block have the same value of x in 8 bits representation. As a first step of the search the relevant block is located, and then the exact pixel is found inside the block.

3) Fill the RGB data in the pixels on the segment between two successive run-length elements by continuously running over the appropriate line in the data source of snaps.

4) Keep a pointer to the last "used" element in the snap data source, so that it will serve as the starting point of the next search.

This method for cutting of snap regions is valid provided that the snap data is ordered, and the set of the rectangles is ordered.

Packing method

The part in which the packing is performed is quite straight forward. The method makes a merge between the data of the clusters and the data of the cut snaps. The packing is typically performed only for clusters for which the associated snap regions already exist. For each rectangular snap the associated cluster is located, and a new sip window (of the size of the snap) is opened. The cluster and the snap are written in the format of sipdata, as sipdata_cluster and sipdata_RGB, respectively, and added to the window. A test function is associated, and the window is added to the queue of windows and it is ready for further processing.

Summary

The task Task_trigger_handler gets as an input two data sources (data source of color defects and data source of snaps), and produces a queue of windows, that contains two sipdata objects - sipdata_cluster, and sipdata_RGB. Sipdata_cluster contains a list of color_defects that belong to a cluster, and some information about the cluster, like its type. Sipdata_RGB contains the RGB pixels inside a rectangular region that contains the color_defects of a particular cluster, ordered by the location, stored as an image in ppm format. The location of the top-left corner of rectangular image is given in an absolute coordinate system.

Several issues may be added or changed in the future, according to specific needs:

- Division into 3 tasks : Clusterer, Snap cutter, Packer.
- Clustering according to a general distance criterion.
- The shape of the cluster may be stored.

An example of the clustering method, provided below, may be understood with reference to Figures 66-67B.